Preprint
UCRL-JC-137238

"Graph Based Multilevel Algorithms for Preconditioning Finite Element Problems"

P. S. Vassilevski

This article was submitted to
6th Copper Mountain Conference on Iterative Methods
Copper Mountain, CO
April 5, 2000

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

March 24, 2000

DISCLAIMER

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

"Graph based multilevel algorithms for preconditioning finite element problems"

by

Panayot S. Vassilevski

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

Sixth Copper Mountain Conference on Iterative Methods
April 5, 2000

---

* Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Content

1. A general block–factorization (matrix) form of multilevel preconditioners; algebraic methods;

2. Selecting parameters based on the matrix topology; graph based algorithms;

3. Examples of coarsening;

4. Numerical experiments;

1. A general block–factorization (matrix) form of multilevel preconditioners; algebraic methods

Consider a sparse matrix A.

Having a transformation matrix $Y = [Y_1,\ Y_2]$ one can introduce block structure in A:

$$\widehat{A} \equiv Y^T A Y = \begin{bmatrix} Y_1^T A Y_1 & Y_1^T A Y_2 \\ Y_2^T A Y_1 & Y_2^T A Y_2 \end{bmatrix}.$$

Given $\widehat{B}_i$ precondtioners to $\widehat{A}_i \equiv Y_i^T A Y_i$, $i = 1, 2$, respectively,

define the following block factorization preconditioner for $\widehat{A}$:

$$\widehat{B} = \begin{bmatrix} \widehat{B}_1 & 0 \\ Y_2^T A Y_1 & \widehat{B}_2 \end{bmatrix}$$
$$\times \begin{bmatrix} (2I - \widehat{B}_1^{-1}\widehat{A}_1)^{-1} & 0 \\ 0 & I \end{bmatrix}$$
$$\times \begin{bmatrix} I & \widehat{B}_1^{-1} Y_1^T A Y_2 \\ 0 & I \end{bmatrix}.$$

The actual preconditioner to A is then defined as:

$$B^{-1} = Y \widehat{B}^{-1} Y^T.$$

The actions of B^{-1} are computed with the following algorithm:

Algorithm 1 *[Block elimination]*

Consider $\mathbf{x} = B^{-1}\mathbf{d} = [Y_1,\ Y_2]\widehat{B}^{-1}\begin{bmatrix} Y_1^T\mathbf{d} \\ Y_2^T\mathbf{d} \end{bmatrix}$.

Denote $\widehat{\mathbf{d}}_i = Y_i^T\mathbf{d},\ i = 1, 2$ *and*

$\widehat{A}_{12} = Y_1^T A Y_2,\ \widehat{A}_{21} = Y_2^T A Y_1.$

- **Forward elimination**:

  1. *Compute* $\widehat{\mathbf{x}}_1^0 = \widehat{B}_1^{-1} Y_1^T \mathbf{d}$;

  2. *Eliminate first block, and solve with* $\widehat{B}_2$,

$$\begin{aligned} \widehat{\mathbf{x}}_2 &= \widehat{B}_2^{-1}(\widehat{\mathbf{d}}_2 - \widehat{A}_{21}\widehat{\mathbf{x}}_1^0) \\ &= \widehat{B}_2^{-1} Y_2^T(\mathbf{d} - A Y_1 \widehat{\mathbf{x}}_1^0) \\ &= \widehat{B}_2^{-1} Y_2^T(\mathbf{d} - A Y_1 \widehat{B}_1^{-1} Y_1^T \mathbf{d}) \\ &= \widehat{B}_2^{-1} Y_2^T(I - A Y_1 \widehat{B}_1^{-1} Y_1^T)\mathbf{d}; \end{aligned}$$

- **Backward recurrence:**

  1. Solve:

$$\widehat{B}_1\hat{\mathbf{x}}_1 + \widehat{A}_{12}\hat{\mathbf{x}}_2 = (2\widehat{B}_1 - \widehat{A}_1)\hat{\mathbf{x}}_1^0;$$

  I.e., compute,

$$\begin{aligned}\hat{\mathbf{x}}_1 &= -\widehat{B}_1^{-1}Y_1^T A Y_2\hat{\mathbf{x}}_2 + 2\hat{\mathbf{x}}_1^0 - \widehat{B}_1^{-1}\widehat{A}_1\hat{\mathbf{x}}_1^0 \\ &= 2\hat{\mathbf{x}}_1^0 - \widehat{B}_1^{-1}Y_1^T A(Y_1\hat{\mathbf{x}}_1^0 + Y_2\hat{\mathbf{x}}_2);\end{aligned}$$

  2. Compute the solution $\mathbf{x} = Y_1\hat{\mathbf{x}}_1 + Y_2\hat{\mathbf{x}}_2$, i.e.,

$$\begin{aligned}\mathbf{x} &= 2Y_1\hat{\mathbf{x}}_1^0 - Y_1\widehat{B}_1^{-1}Y_1^T A\left(Y_1\hat{\mathbf{x}}_1^0 + Y_2\hat{\mathbf{x}}_2\right) \\ &\quad + Y_2\hat{\mathbf{x}}_2 \\ &= Y_1\hat{\mathbf{x}}_1^0 + (I - Y_1\widehat{B}_1^{-1}Y_1^T A)(Y_1\hat{\mathbf{x}}_1^0 + Y_2\hat{\mathbf{x}}_2).\end{aligned}$$

One has,

$$\mathbf{r} = \mathbf{d} - AB^{-1}\mathbf{d} = \left(I - AY_1\widehat{B}_1^{-1}Y_1^T\right) \times \left(I - AY_2\widehat{B}_2^{-1}Y_2^T\right)\left(I - AY_1\widehat{B}_1^{-1}Y_1^T\right)\mathbf{d}. \quad (1)$$

This is a "two–level" algorithm of product form.

The trivial choice of $Y_2 = \begin{bmatrix} 0 \\ I \end{bmatrix} \begin{matrix} \} \text{ fine dofs} \\ \} \text{ coarse dofs} \end{matrix}$ corresponds to (approximate) block–factorization preconditioners.

One can view $\widehat{B}_1$ as smoother, Y_2^T as restriction and Y_2 as interpolation. Then, $\widehat{A}_2 = Y_2^T A Y_2$ is the coarse–grid matrix.

Y_1 specifies the actual space where the smoothing is performed.

When $\widehat{B}_2$ is constructed from $\widehat{A}_2$ using the same algorithm (by recursion) one ends up with multilevel methods.

We refer to "algebraic" multilevel methods when the parameters Y_1, Y_2 and $\widehat{B}_1$ are algebraically constructed, i.e., depending on A only.

Typical choices of Y_1 are:

- $Y_1 = \begin{bmatrix} I \\ 0 \end{bmatrix} \begin{matrix} \} \text{ fine dofs} \\ \} \text{ coarse dofs} \end{matrix}$

  – this is HB (hiearchical basis)–like;

- $Y_1 = I$ (both coarse and fine dofs)
  – this is AMG;

- $Y_1 = (I - \pi) \begin{bmatrix} I \\ 0 \end{bmatrix} \begin{matrix} \} \text{ fine dofs} \\ \} \text{ coarse dofs} \end{matrix}$

  – this is the "wavelet"–modified HB;

  here $\pi = Y_2 \tilde{G}_2^{-1} Y_2^T G$ is a an approximate projection on the coarse–grid, G is a given Gram matrix, $G_2 = Y_2^T G Y_2$–the coarse one, and $\tilde{G}_2^{-1} \approx G_2^{-1}$.

2. Selecting parameters based on matrix topology; graph based algorithms;

In what follows we will concentrate on the construction of Y_2 - the coarse–to–fine interpolation matrix, which involves:

the choice of the coarse dofs $\mathcal{D}_k$,

and the interpolation rule P_k itself.

In AMG none of the following is explicitly known:

neither the geometry (domain) and the coefficients of the PDE, nor the elements and type of basis functions. In conventional AMG one assumes that only the assembled sparse matrix A is explicitly given and one has access to its individual entries.

2.1. Main assumptions in AMGe

The main question is how to construct P_k (and $\mathcal{D}_k$) based on the minimal information AMG assumes about the original problem ?

In the conventional AMG (Ruge and Stüben), one is motivated by finite difference M–matrices.

In a general finite element setting, however, the matrices A are seldom M–matrices.

In 1998, by a team from University of Colorado, Boulder and LLNL, the following assumption was made to generalize AMG to finite element problems, called **AMGe**; namely, that one allows

"**access to the individual element matrices**" (on the fine grid).

This assumption can be eliminated (to a certain extend) but in the present talk we will use it.

2.2. Main definitions and constructions

By definition, an element is a "list of degrees of freedom", $e = \{d_1, \ldots, d_{n_e}\}$,

and we are given an overlapping partition $\{e\}$ of $\mathcal{D}$ (the set of degrees of freedom).

Also, each element is associated with an element matrix A_e, an $n_e \times n_e$ matrix, which in our application we assume to be symmetric and positive semi–definite.

Then, the global matrix A is assembled from the individual element matrices A_e in the usual way, i.e.,

$$\mathbf{w}^T A \mathbf{v} = \sum_e \mathbf{w}_e^T A_e \mathbf{v}_e.$$

Here, $\mathbf{v}_e = \mathbf{v}|_e$, i.e., restriction to subset ($e \subset \mathcal{D}$).

2.3. Graph based algorithms for coarsening

The variant of AMGe method we will present is based on a straightforward extension of the standard finite element method, now in a graph setting.

Namely, given the following graph,

"**element_node**"

which is the incidence sparse matrix "element" i (rows) contains "node" j (columns),

i.e., it is the rectangular sparse matrix

Element_Node of ones and zeros, of size

(number of elements) $\times$ (number of nodes).

("node" means a number of degrees of freedom in order to handle systems of PDEs).

The incidence

"node" i belongs to "element" j,

is given by the transpose of the above rectangular sparse matrix, i.e.,

Node_Element = (Element_Node)^T.

One can consider a number of useful graphs (easily computable)

"**element_element**" =

= "**element_node**" $\times$ "**node_element**",

"**node_node**" = "**node_element**" $\times$ "**element_node**",

The first one shows the incidence

"element" i intersects "element" j,

whereas the second one shows the sparsity pattern of the assembled matrix, namely,

"node" i is connected to "node" j (hence the entry $a_{i,j}$ could be non–zero).

2.1. Element faces

The notion of "**face**" (similar to standard elements) is defined as a maximal intersection set, i.e., consider all intersections

$e_1 \cap e_2$, $e_1 \neq e_2$.

A face is a maximal intersection set of the above type, or a maximal intersection set of the type

$e \cap$ "boundary surface".

(if special lists of nodes is given, that provide additional information about the domain boundary)

One can then construct the following graphs:

"**element_face**", "**face_element**", "**face_node**", "**face_face**", etc.

For example,

"**face_face**" = "**face_node**" $\times$ "**node_face**".

2.2. Element agglomeration

The topological information is used to devise an algorithm to agglomerate elements – a new overlapping partition $\{E\}$ of $\mathcal{D}$ where each $E = e_1 \cup e_2 \ldots \cup e_p$, i.e., to build the new graph

"**AE_element**" where **AE** stands for "agglomerated element".

The following algorithm has been proposed in Jones and V. (1999).

The motivation is to have "quasiuniform" "AE"s. In particular, this algorithm will restore coarse rectangular or triangular elements (up to boundary effects).

Algorithm 2 (Agglomeration of elements)

Given the graphs

*"***face_face***", "***element_face***" and "***face_element***"*

and a weight function $w(f) = 0$, f-face, one performs:

1. *find a face f with maximal $w(f) \geq 0$, then set $w(f) = -1$ and add on the list of the current "AE" the elements e_1 and e_2 such that $f = e_1 \cap e_2$.*

2. *update $w(f_1)$ for all f_1 connected to f (based on the graph "***face_face***"), according to the following topological rule, $w(f_1) := w(f_1) + 1$ if f_1 is connected to f and once more $w(f_1) := w(f_1) + 1$ if f_1 and f belong to a same element (here one uses the graph "***face_element***");*

3. *if for all faces f_1 of the already agglomerated elements e (here one uses the graph "***element_face***") in the current "AE" $w(f_1)$ is less than $w(f)$ where f was the last eliminated face, the agglomeration procedure for the current "AE" is terminated. Then, go to step 1 or stop.*

2.3. Selection of coarse nodes

Assume that the graph "**AE_element**" has been constructed (somehow), then one can build

"**AE_face**" = "**AE_element**" × "**element_face**".

Finally, one can define faces of agglomerated elements, "AEface"s, based on "**AE_face**"; namely, each "AE" has a number of faces of the original elements. That is, we have the lists "AE" – "faces". Intersecting two different lists, one gets the faces of the "AE"s in terms of the faces of the original elements. That is, one may define the new graphs

"**AEface_face**", and "**AE_AEface**".

2.4. Vertices – nodes that belong to two (or more) "AEface"s. For this we need the graph

$$\text{"node\_AEface"} = (\text{"AEface\_node"})^T,$$

which can be computed as the transpose of

$$\text{"AEface\_node"} = \text{"AEface\_face"} \times \text{"face\_node"}.$$

Definition 1 (Coarse nodes) *A minimal set of coarse nodes $\mathcal{N}_c \subset \mathcal{N}$ is provided by the vertices of the "AE"s, i.e., one forms the graph* "node_coarsenode";

Then one can construct,

"coarseelement_coarsenode" =

"AE_node" × "node_coarsenode";

"coarseelement_coarseface" = "AE_AEface";

"coarseface_coarsenode" =

"AEface_node" × "node_coarsenode";

and hence be able to continue coarsening by recursion.

2.5. Construction of interpolation mappings

Having chosen a coarse grid $\mathcal{N}_c$ one can construct interpolation mappings based on "minimal energy" principle. I.e., given a node d one needs a set of coarse nodes to interpolate from. This can be based on the topology already created. Given the graphs,

"**AE_node**" = "**AE_element**" $\times$ "**element_node**",

and its transpose "**node_AE**",

one can build a neighborhood $\Omega(d) = \cup\{E,\ d \in E\}$ of d. I.e., from the list "node"– "agglomerated element", one forms the union of all "AE"s that contain d.

Then assemble the stiffness matrix $A_{\Omega(d)}$ from the element matrices associated with the elements e that are contained in "AE"s in $\Omega(d)$.

In order to get a compatible interpolation rule one uses (Jones and V. (1999)):

"**A node d is interpolated only from coarse nodes that belong to** $N(d) = \cap\{E,\ E \in \Omega(d)\}$".

Then, one can define coarse element matrices, as

$$A^c_{E_c} = (P|_E)^T A_E\, P|_E\,.$$

Here, A_E is the assembled matrix corresponding to each "AE", $E = e_1 \cup e_2 \ldots \cup e_p$, and P is restricted to E. Note that for each E, fine–grid nodes ($\in E$) are interpolated from coarse nodes from E, hence the restriction of P to each E is well–defined.

Thus, the entire coarsening procedure can be recursively applied.

Finally, one may separate the graph algorithms for

generating agglomerated elements

from the

selection of the coarse grids.

One may define the coarse-grid nodes **independently** of the "AE"s.

In that case, however, one has to **give up** the property

that the global coarse matrix A_c can be assembled from the coarse element matrices.

It is **desirable** to have $A_c = P^T A P$.

The idea is to have **two sets** of matrices,

"true" stiffness matrices,

related variationally,

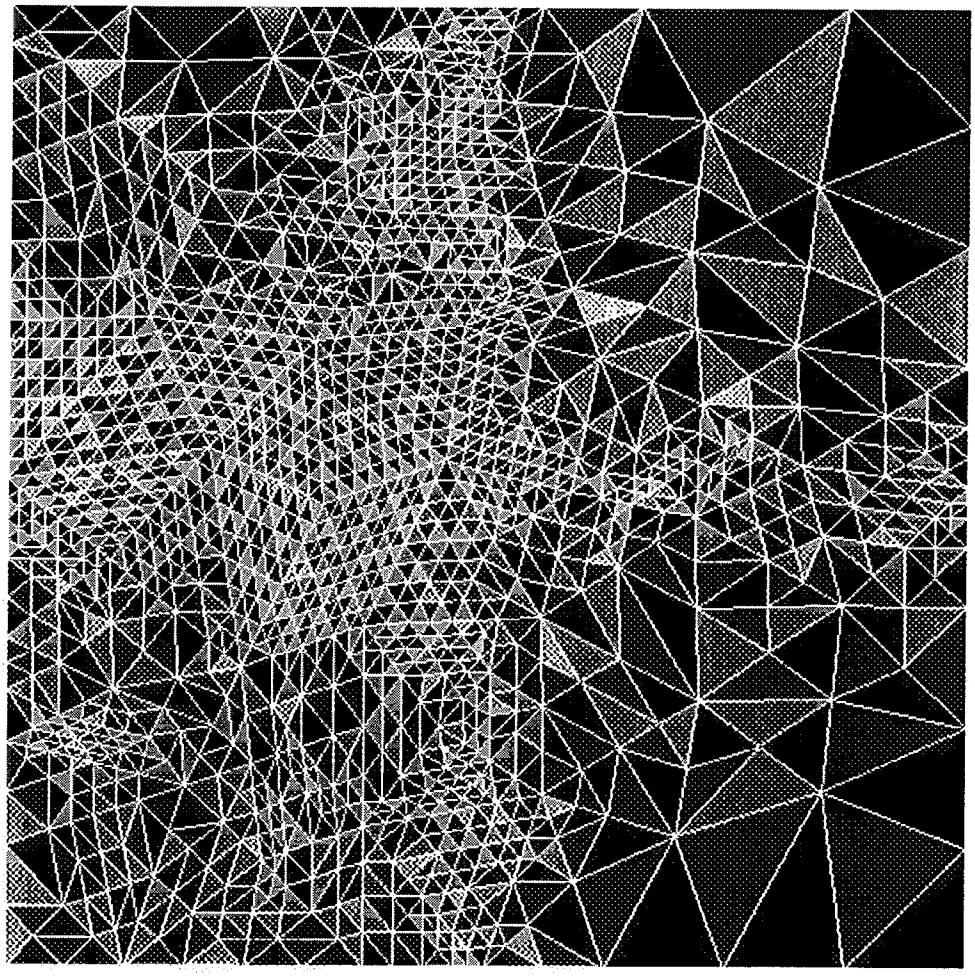
and *element matrices* that are only needed as a tool to build P.

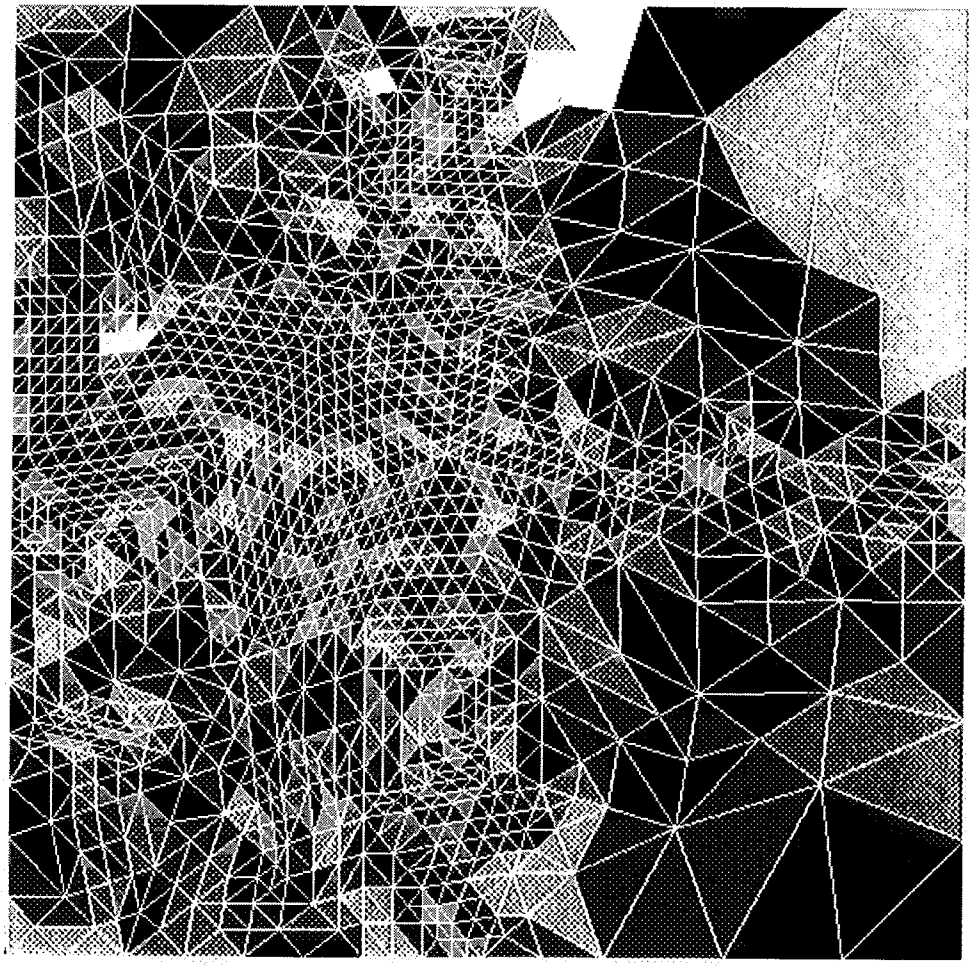
The latter can be viewed as "non–conforming" coarse discretization element matrices. They are defined as before, i.e., "AE"–wise,

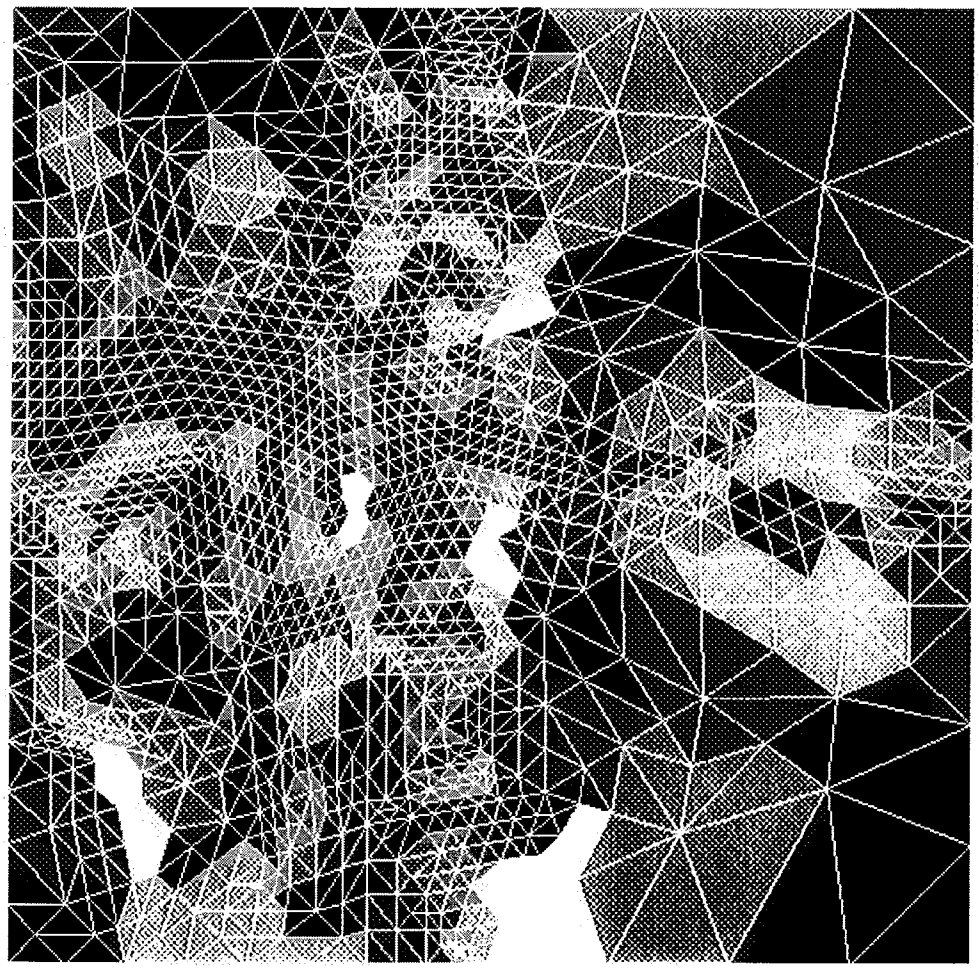
$$A^c_{E_c} = (P_E)^T A_E P_E;$$

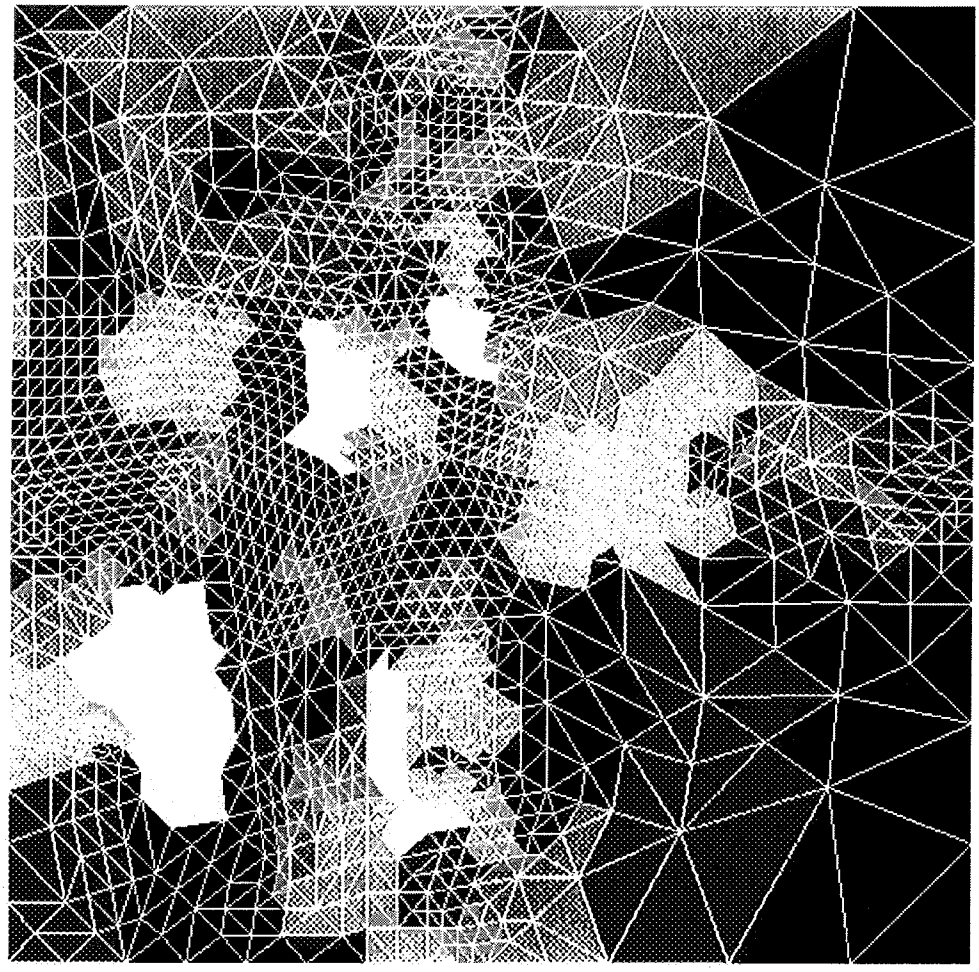
however, $\{P_E\}$ does not necessarily define a global prolongation mapping, i.e., it is not, in general, a compatible set of prolongation mappings.

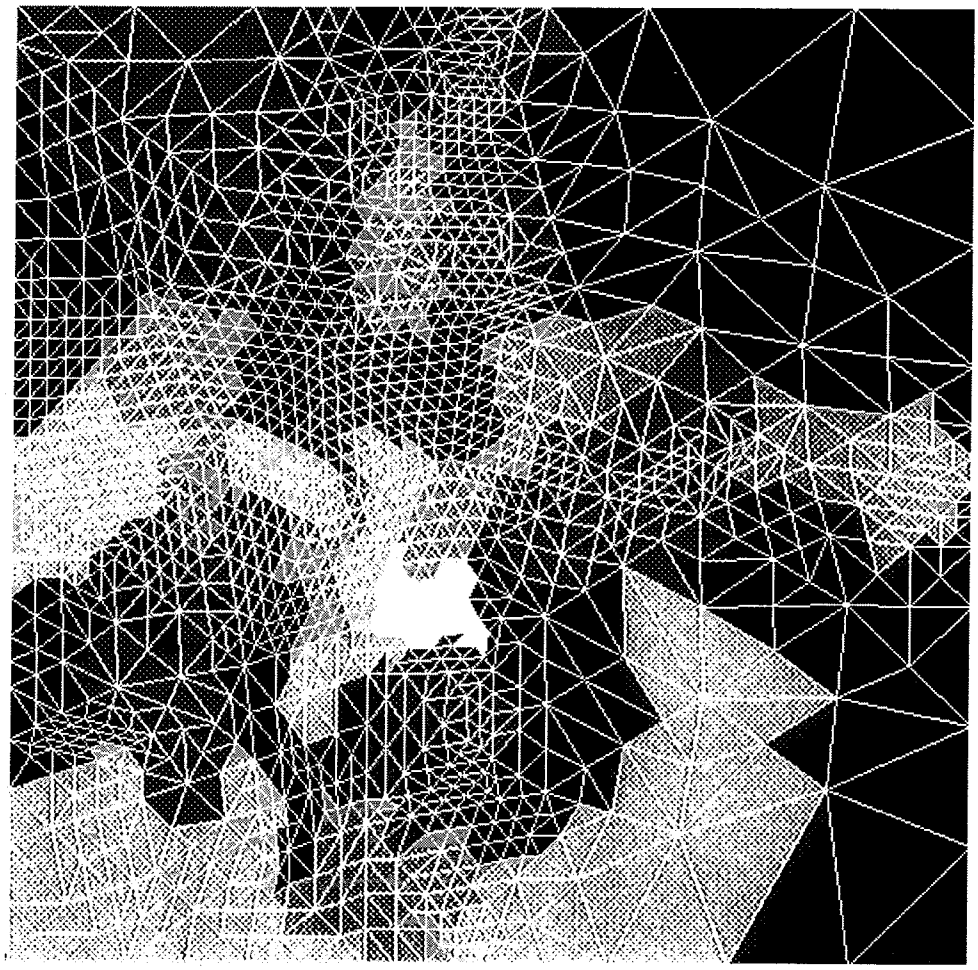
3. Examples of coarse elements

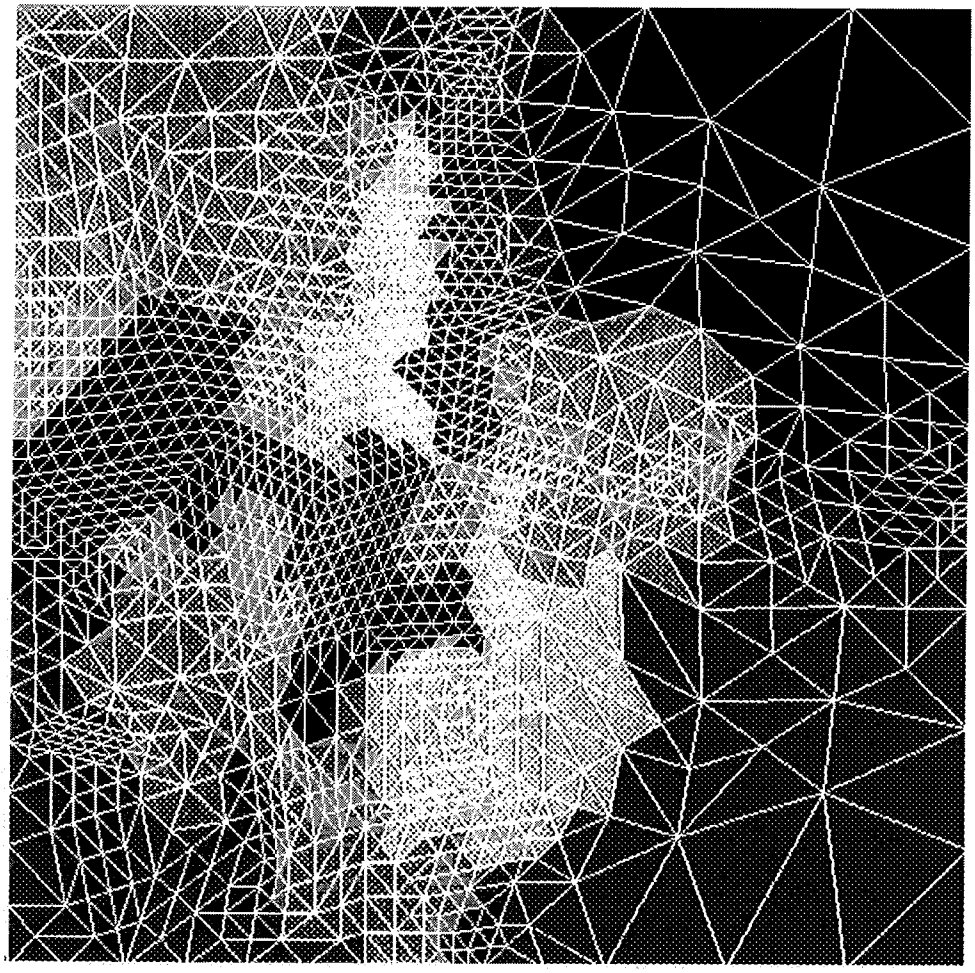


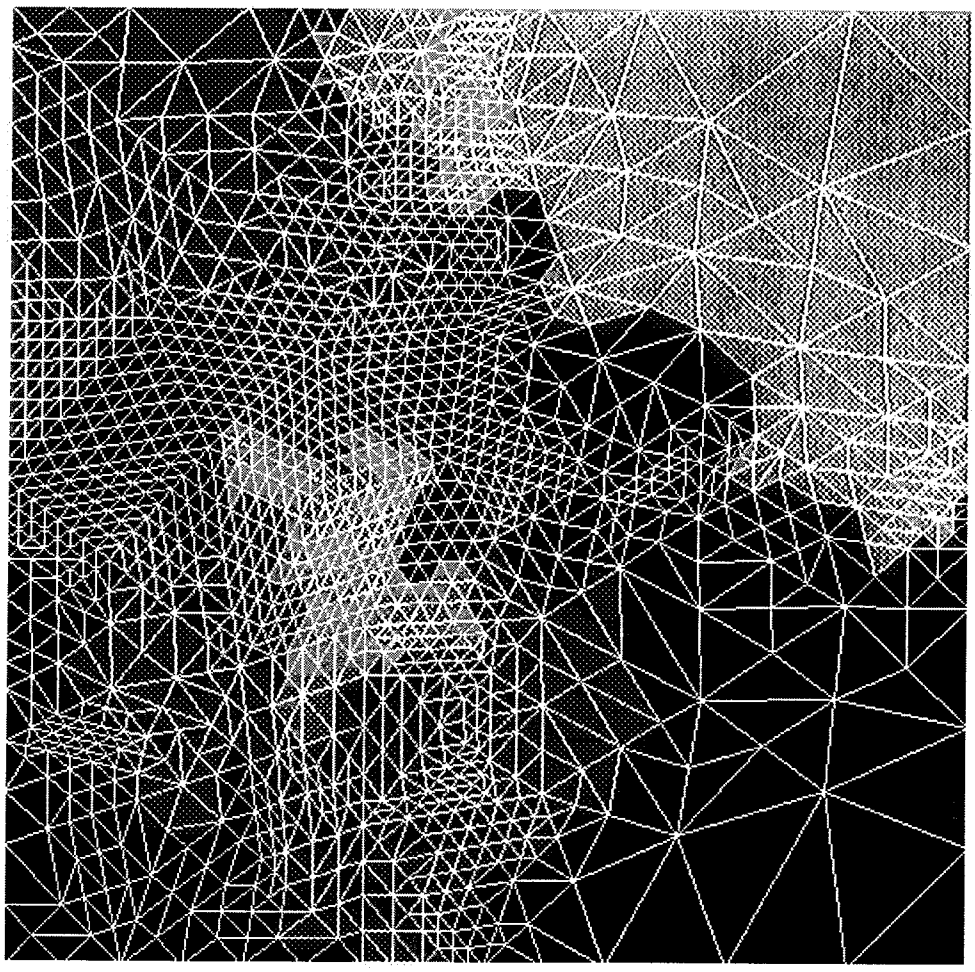


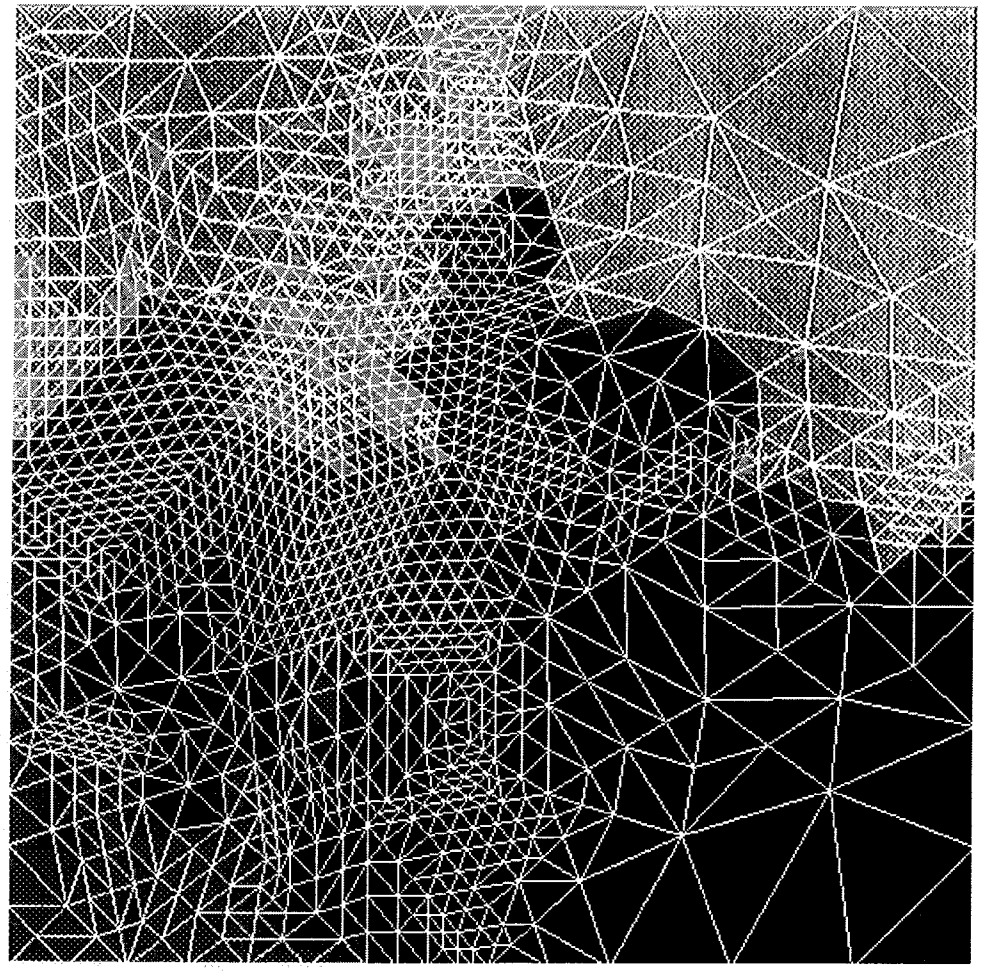


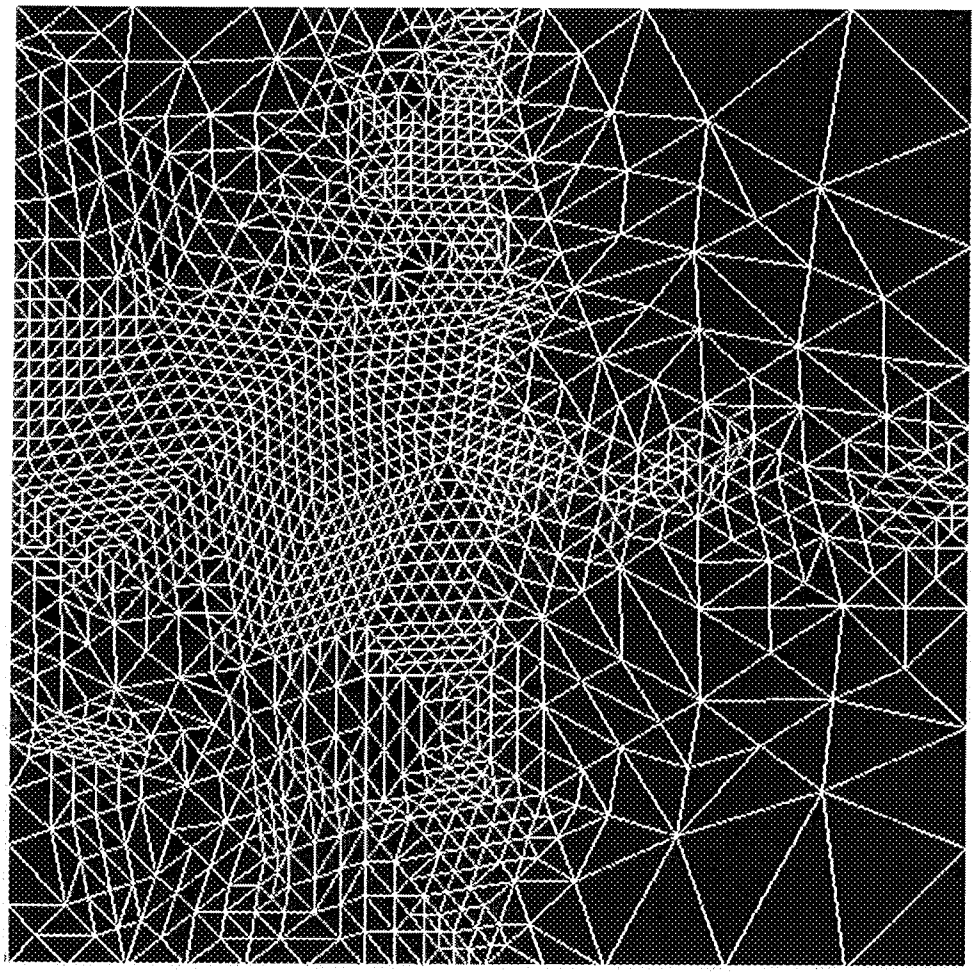






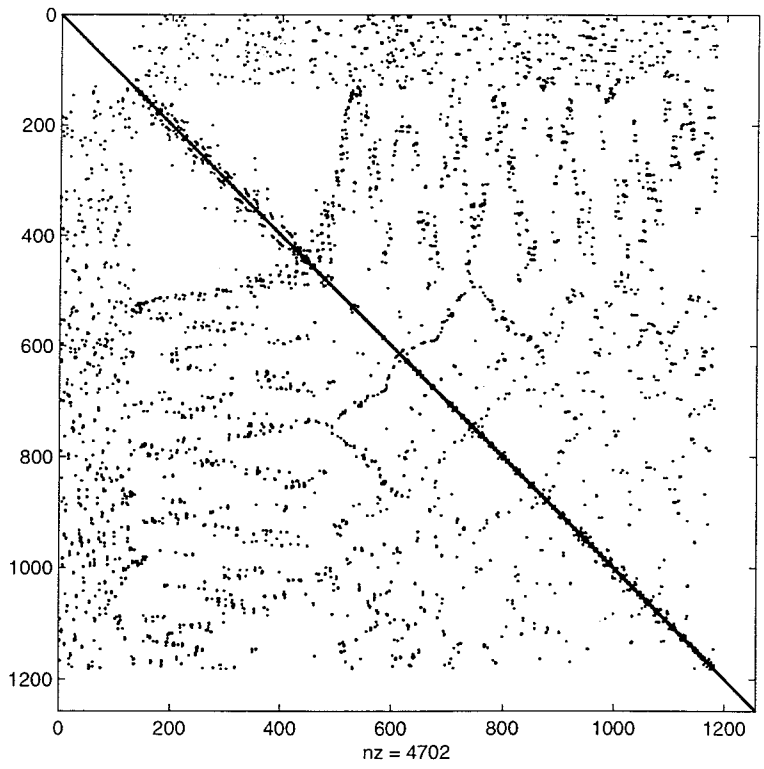


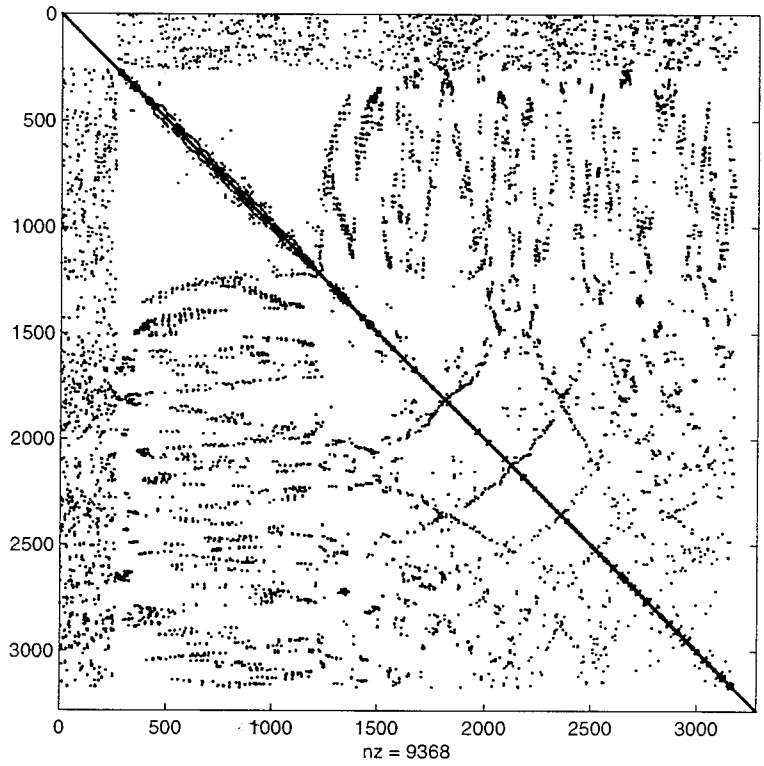


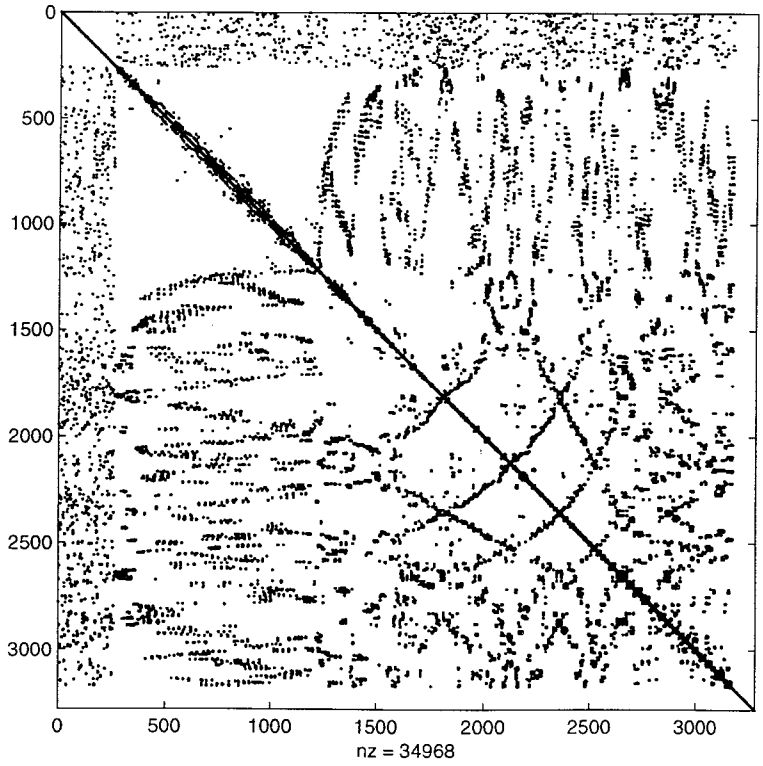


The graph-based agglomeration algorithm is a partitioning algorithm and provides natural nested dissection ordering of the matrices.

Typical sparsity patterns are as follows:




0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
nz = 9368


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000
2500
3000
nz = 34968

PCG tests with approcimate LU–factorization of the matrix in the nested dissection ordering; sparsity pattern of the preconditioner is determined by the sparsity pattern of A^2, i.e., a ILU(1)–preconditioner;

Table 1 *Iteration counts and convergence factors for AMGe; unstructured triangular grid, anisotropic Poisson equation; # fine elements = 6 452, # fine dofs = 3 281, # levels = 9*

method	*ILU*	*V-cycle –ILU smoother*	*PCG –GS smoothe*
# iterations	*48*	*30*	*65*
ϱ	*0.75*	*0.42*	*0.57*

Table 2 *Iteration counts and convergence factors for AMGe; unstructured triangular grid, anisotropic Poisson equation; # fine elements = 2 435, # fine dofs = 1 256, # levels = 8*

method	*ILU*	*V-cycle –ILU smoother*	*PCG –GS smoothe*
# iterations	*24*	*21*	*48*
ϱ	*0.46*	*0.29*	*0.57*

Table 3 *Iteration counts and convergence factors for AMGe; unstructured triangular grid, anisotropic Poisson equation; # fine elements = 876, # fine dofs = 466, # levels = 7*

method	*ILU*	*V-cycle –ILU smoother*	*PCG –GS smoother*
# iterations	*12*	*12*	*36*
ϱ	*0.3*	*0.12*	*0.46*

Table 4 *Iteration counts and convergence factors for AMGe; unstructured triangular grid, anisotropic Poisson equation; # fine elements = 296, # fine dofs = 169, # levels = 6*

method	*ILU*	*V-cycle –ILU smoother*	*PCG –GS smoothe*
# iterations	*6*	*5*	*22*
ϱ	*0.05*	*0.007*	*0.27*

4. Numerical experiments

"Standard examples": quasiuniform meshes and Poisson like problems;

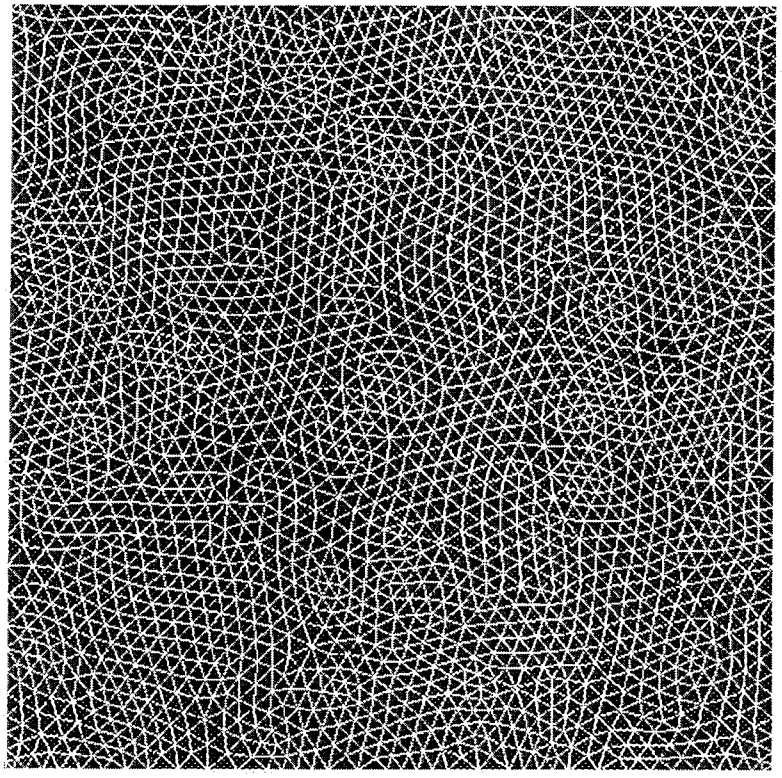


Table 5 *Iteration counts and convergence factors for AMGe; unstructured triangular grid, Poisson equation.*

# elements	*4 016*	*16 016*
# dofs	*2 085*	*8 095*
# levels	*8*	*10*
# iterations	*14*	*17*
ϱ	*0.13*	*0.18*

Table 6 *AMGe performance for 2–D elasticity; non–conforming coarse element matrices used and graph (nodal) coarsening applied;* $\mu = 1,\ \lambda = 1.$

h^{-1}	d	# levels	# iterations	ϱ	# unknowns	# element
16	1	5	17	0.18	578	256
32	1	6	18	0.21	2178	1024
64	1	7	21	0.26	8450	4096
128	1	8	24	0.30	33282	16384
128	0.125	8	26	0.34	4386	2048
128	0.25	8	23	0.29	8514	4096

Table 7 *Typical coarsening history; unstructured triangular grid.*

level #	*# elements*	*# nodes*	*matrix size*	*min matrix siz*
0	*4016*	*2085*	*14285*	*14285*
1	*866*	*917*	*10405*	*9055*
2	*240*	*395*	*8383*	*4521*
3	*95*	*177*	*5829*	*2031*
4	*43*	*84*	*3456*	*960*
5	*20*	*42*	*1462*	*432*
6	*9*	*20*	*394*	*166*
7	*3*	*8*	*64*	*46*
8	*1*	*4*	*16*	*16*

"Non–standard example":

$$-\nabla \cdot (\epsilon I + \underline{b}\underline{b}^T)u = 1, \quad \text{in } \Omega = (0,1)^2, \qquad (2)$$

where $\underline{b} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$.

$$\theta = \begin{cases} 0, & 0 < x < \frac{1}{2},\ \frac{1}{2} < y < 1, \\ \frac{\pi}{4}, & \frac{1}{2} < x < 1,\ \frac{1}{2} < y < 1, \\ -\frac{\pi}{4}, & \frac{1}{2} < x < 1,\ 0 < y < \frac{1}{2}, \\ \frac{\pi}{2}, & 0 < x < \frac{1}{2},\ 0 < y < \frac{1}{2}; \end{cases}.$$

and

$$\epsilon = 0.001.$$

Here we use as coarse grid all nodes on the faces of the agglomerated elements plus some additional interior nodes which make the corresponding A_{ff} block of the matrices better–conditioned.

This an extreme case of coarsening and can be viewed as an approximate "nested dissection" type factorization of the matrix plus standard (Gauss–Seidel) smoothing.

The coarse matrices are getting denser and the cost is high, but the resulting method is robust with respect to highly anisotropic coeffcients and standard quasiuniform fine meshes.

Table 8 *Iteration counts and convergence factors for AMGe; unstructured triangular grid, anisotropic Poisson equation.*

# elements	296	876	2 435	6 452
# dofs	169	466	1 256	3281
# levels	6	7	8	9
# iterations	11	13	16	21
ϱ	0.07	0.10	0.17	0.26

Table 9 *Typical coarsening history; unstructured triangular grid.*

level #	*# elements*	*# nodes*	*matrix size*
0	*6452*	*3281*	*22745*
1	*1408*	*3023*	*25947*
2	*402*	*2072*	*46522*
3	*146*	*1475*	*59209*
4	*65*	*1165*	*76805*
5	*31*	*922*	*92128*
6	*15*	*762*	*128950*
7	*7*	*631*	*191043*
8	*3*	*459*	*171213*
9	*1*	*108*	*11664*

Table 10 *Typical coarsening history; unstructured triangular grid.*

level #	*# elements*	*# nodes*	*matrix size*
0	*2435*	*1256*	*8636*
1	*502*	*1129*	*10365*
2	*145*	*805*	*18673*
3	*62*	*645*	*25215*
4	*28*	*508*	*31636*
5	*13*	*390*	*38126*
6	*6*	*314*	*47404*
7	*3*	*257*	*52921*
8	*1*	*75*	*5625*

University of California
Lawrence Livermore National Laboratory
Technical Information Department
Livermore, CA 94551